Conceptual XML for Systems Analysis

A Dissertation Proposal
Presented to the
Department of Computer Science
Brigham Young University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Reema Al-Kamha
March 2005

1 Introduction

XML is rapidly becoming the standard for data representation, especially for data that is exchanged on the web. XML Schema [XML01] is used to describe the structure and the content of XML data. Although XML Schema is useful for specifying and validating XML documents, systems analysts who store their models using XML need a simple conceptual model to help improve modeling capabilities for XML-based development. This need for conceptual modeling arises because of the drawbacks of XML Schema. XML Schema overexposes analysts to low-level, implementation details, and the structure of XML Schema forces analysts to view all data hierarchically, even when its natural structure is not hierarchical. Furthermore, XML Schema has only a textual representation; since the early 70's, systems analysts have used graphical versions of conceptual models to aid in understanding and documenting essential characteristics of systems. This capability should be available to systems analysts who store their models using XML as well.

In this proposal we present *Conceptual-XML* (*C-XML*) which meets this new need of systems analysts who store their model using XML. C-XML is first and foremost a conceptual model, being fundamentally based on object-set and relationship-set constructs. Secondly, C-XML is "model-equivalent" [LEW00] with XML Schema, which means that C-XML can represent each component and constraint in XML Schema and vice versa. Being model-equivalent makes it possible to indirectly (but precisely) view, manipulate, and work with an XML schema by directly viewing, manipulating and working with C-XML. Since C-XML represents its model at a much higher level of abstraction and with fewer constraints on its structure, this is a significant advantage. As an example of this advantage, we use this model-equivalence to help solve the difficult problem of integrating XML repositories.

The only other conceptual model we are aware of for XML[1] is the ORA-SS data model [DWLL00]. The designers of ORA-SS have several objectives for their work: (1) translate XML documents to ORA-SS schema diagrams [CLL02], (2) design valid XML views [CLL02], (3) automatically generate XQuery view definitions from views that are defined using the ORA-SS conceptual model [CLL03], (4) do integration between ORA-SS diagrams [YLL03]. Our proposed research and ORA-SS research have some major similarities. First, both have a conceptual model (C-XML in our proposed work and ORA-SS in their work). Second, both make use of conceptual modeling to assist in the integration process. Our proposed research and ORA-SS research also have important differences. First, ORA-SS has a hierarchial structure, while C-XML has an unconstrained structure. Second, in the work on ORA-SS, the transformation is only done from an XML document to an ORA-SS conceptual model instance. Thus, the translation is only for documents and is only in one direction. In our proposed work, we can do the translation for both documents and schemas and in both directions, from C-XML to XML and also from XML to C-XML. Third,

[1] As we were completing this proposal, we became aware that BEA [BEA] has undertaken an initiative to create a conceptual model for XML. We will follow their developments as we carry out this work.

the translation from an XML document to an ORA-SS model instance needs semantic enrichment, while in our proposed work we directly translate from an XML schema to C-XML, taking all the constraints from the XML schema and thus requiring no enrichment. Fourth, in the integration work, the ORA-SS work only makes use of the transformation from an XML document to an ORA-SS conceptual model instance, while in our proposed work, in addition to the transformation from XML Schema to C-XML, we also translate from C-XML to XML Schema, and thus we obtain an integrated XML schema.

2 Thesis Statement

Because XML has become a new standard for data representation, there is a need to produce a simple conceptual model that works well with XML-based development. In this proposal, we present Conceptual-XML (C-XML) which meets this new need of systems analysts who store their model using XML. In addition, we propose the following: (1) mappings to and from C-XML and XML Schema that preserve information and constraints and (2) integration of C-XML model instances resulting in the integration of the underlying, equivalent XML schemas. Further, we propose to mathematically prove the correctness of the transformations included within C-XML/XML Schema mappings and the integration of C-XML model instances and XML schemas. A prototype implementation will demonstrate the practicality of the theory.

3 Research Description

3.1 C-XML: Conceptual XML

C-XML is a conceptual model consisting of object sets, relationship sets, and constraints over these object and relationship sets. In addition, with each object set we associate a data frame to provide a rich description of its value set and other properties. Graphically a C-XML model instance M is an augmented hypergraph whose vertices and edges are respectively the object sets and relationship sets of M, and whose augmentations consist of decorations that represent constraints. Figure 1 shows an example. Although based on [EKW92], the particular notation we use to represent C-XML is not significant. The hypergraph foundation, however, is significant because it is more directly amenable to the requirements of XML and XML Schema. Thus, this choice simplifies translations, abstraction definitions, and conceptual model instance merging.

3.2 Translations between C-XML and XML Schema

Many translations between C-XML and XML Schema are possible. In recent ER conferences, for example, researchers have described varying conceptual-model translations to and/or from XML or XML DTD's or XML-Schema-like specifications, [BGH00, CSF00, MLM01, dH01, EM01, EWH$^+$02, CLL02]. It is not our purpose here to argue for or against a particular translation. Indeed, we would

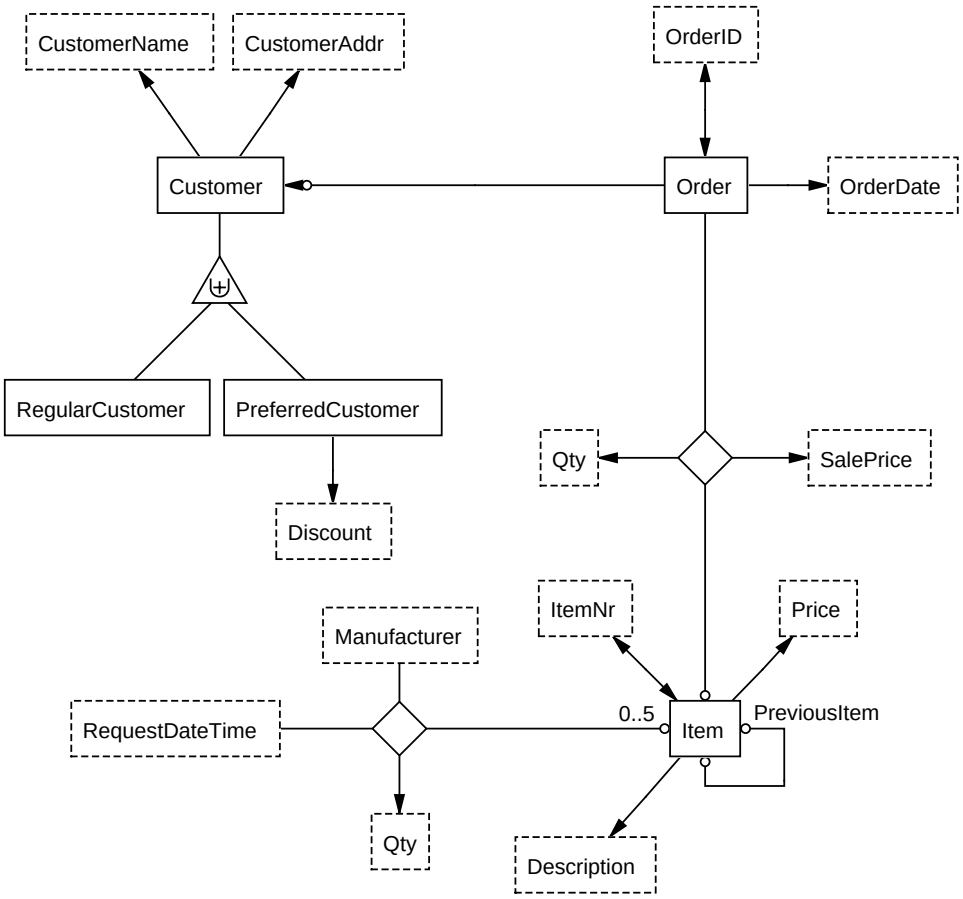

Figure 1: Customer/Order C-XML Model Instance.

argue that a variety of translations may be desirable. For any translation, however, we require information and constraint preservation. This ensures that an XML schema and a conceptual instantiation of an XML schema as a C-XML model instance correspond and that a system can reflect manipulations of the one in the other.

To make our correspondence exact, we need information- and constraint-preserving translations in both directions. We do not, however, require that translations be inverses of one another—translations that generate members of an equivalence class of XML Schema specifications and C-XML model instances are sufficient. In Section 3.2.1 we outline a possible C-XML-to-XML-Schema translation, and in Section 3.2.2 we outline a possible XML-Schema-to-C-XML translation.

3.2.1 Translation from C-XML to XML Schema

We illustrate our proposed translation process by showing the C-XML model instance of Figure 1 translated to the corresponding XML schema of Figure 2. Our approach to translation from C-XML model instance C to an XML schema S_C is based on [EM01]. For our example in Figure 1, the algorithms in [EM01] will generate the following two nested scheme trees.

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 3:            elementFormDefault="qualified" attributeFormDefault="unqualified">
 4: <xs:element name="Document">
 5:     <xs:complexType>
 6:         <xs:choice minOccurs="0" maxOccurs="unbounded">
 7:             <xs:element ref="Customer"/>
 8:             <xs:element name="Item">
 9:                 <xs:complexType>
10:                     <xs:sequence>
11:                         <xs:element name="ItemMR" minOccurs="0" maxOccurs="5">
12:                             <xs:complexType>
13:                                 <xs:attribute name="Manufacturer" type="xs:string" use="required"/>
14:                                 <xs:attribute name="RequestDateTime" type="xs:date" use="required"/>
15:                                 <xs:attribute name="Qty" type="xs:positiveInteger" use="required"/>
16:                             </xs:complexType>
17:                         </xs:element>
18:                         <xs:element name="PreviousItem" minOccurs="0" maxOccurs="unbounded">
19:                             <xs:complexType>
20:                                 <xs:attribute name="ItemNr" type="xs:positiveInteger" use="required"/>
21:                             </xs:complexType>
22:                             <xs:keyref name="r1" refer="ItemKey">
23:                                 <xs:selector xpath="."/>
24:                                 <xs:field xpath="@ItemNr"/>
25:                             </xs:keyref>
26:                         </xs:element>
27:                     </xs:sequence>
28:                     <xs:attribute name="ItemNr" type="xs:positiveInteger" use="required"/>
29:                     <xs:attribute name="Description" type="xs:string" use="required"/>
30:                     <xs:attribute name="Price" type="xs:decimal" use="required"/>
31:                 </xs:complexType>
32:             </xs:element>
33:         </xs:choice>
34:     </xs:complexType>
35:     <xs:key name="OrderKey">
36:         <xs:selector xpath=".//Order"/>
37:         <xs:field xpath="@OrderID"/>
38:     </xs:key>
39:     <xs:key name="ItemKey">
40:         <xs:selector xpath=".//Item"/>
41:         <xs:field xpath="@ItemNr"/>
42:     </xs:key>
43: </xs:element>
44: <xs:element name="Customer" abstract="true"/>
45: <xs:element name="PreferredCustomer" substitutionGroup="Customer">
46:     <xs:complexType>
47:         <xs:group ref="CustomerDetails"/>
48:         <xs:attribute name="Discount" type="xs:string" use="required"/>
49:     </xs:complexType>
50: </xs:element>
51: <xs:element name="RegularCustomer" substitutionGroup="Customer">
52:     <xs:complexType>
53:         <xs:group ref="CustomerDetails"/>
54:     </xs:complexType>
55: </xs:element>
56: <xs:group name="CustomerDetails">
57:     <xs:sequence>
58:         <xs:element name="CustomerName" type="xs:string"/>
59:         <xs:element name="CustomerAddr" type="xs:string"/>
60:         <xs:element name="Order" minOccurs="0" maxOccurs="unbounded">
61:             <xs:complexType>
62:                 <xs:sequence>
63:                     <xs:element name="OrderItem" minOccurs="0" maxOccurs="unbounded">
64:                         <xs:complexType>
65:                             <xs:attribute name="Qty" type="xs:positiveInteger" use="required"/>
66:                             <xs:attribute name="SalePrice" type="xs:decimal" use="required"/>
67:                             <xs:attribute name="ItemNr" type="xs:positiveInteger" use="required"/>
68:                         </xs:complexType>
69:                         <xs:keyref name="r3" refer="ItemKey">
70:                             <xs:selector xpath="."/>
71:                             <xs:field xpath="@ItemNr"/>
72:                         </xs:keyref>
73:                     </xs:element>
74:                 </xs:sequence>
75:                 <xs:attribute name="OrderID" type="xs:positiveInteger" use="required"/>
76:                 <xs:attribute name="OrderDate" type="xs:date" use="required"/>
77:             </xs:complexType>
78:         </xs:element>
79:     </xs:sequence>
80: </xs:group>
81: </xs:schema>
```

Figure 2: XML Schema for the C-XML Model Instance in Figure 1.

$(Customer, CustomerName, CustomerAddr, Discount$
$(Order, OrderID, OrderDate,$
$(Item, SalePrice, Qty)*)*)*$

$(Item, ItemNr, Description, Price,$
$(PreviousItem)*, (Manufacturer, RequestDateTime, Qty)*)*$

The XML schema in Figure 2 satisfies these nesting specifications. Observe that *Item* in the second scheme tree appears as an element in Line 8 with *ItemNr*, *Description*, and *Price* defined as its attributes on Lines 28–30. *PreviousItem* is nested, by itself, underneath *Item*, in Line 18, and *Manufacturer*, *RequestDateTime*, and *Qty* are nested underneath *Item* as a group in Lines 13–15. The XML Schema notation that accompanies these C-XML object-set names obscures the nesting to some extent, but, as we explain in our continuing discussion, this additional notation is necessary either to satisfy the syntactic requirements of XML Schema or to allow us to specify the constraints of the C-XML model instance.

In the conversion from C-XML to XML Schema we use attributes instead of elements where possible. An object set can be represented as an attribute of an element if it is lexical [EKW92], is functionally dependent on the element, and has no order annotations. The object sets *OrderID* and *OrderDate*, for example, satisfy these conditions and appear as attributes of an *Order* element in Lines 75 and 76.

When an object set is lexical but not functional and order constraints do not hold, the object set becomes an element with minimum and maximum participation constraints. *PreviousItem* in Line 18 has a minimum participation constraint of 0 and a maximum of *unbounded*.

Because XML Schema will not let us directly specify n-ary relationship sets ($n > 2$), we convert them all to binary relationship sets by introducing a tuple identifier. To obtain a name for the object set containing the tuple identifiers, we concatenate names of nonfunctionally dependent object sets. For example, given the n-ary relationship set for *Order*, *Item*, *SalePrice*, and *Qty*, we generate an *OrderItem* element (Line 63).

When a lexical object set has a one-to-one relationship with a nonlexical object set, we use the lexical object set as a surrogate for the nonlexical object set and generate a key constraint. In our example, this generates key constraints for *Order*/*OrderID* in Lines 35–38. We also use these surrogate identifiers, as needed, to maintain explicit referential integrity. We generate *keyref* constraints, one in Lines 69–72 to ensure the referential integrity of *ItemNr* in the *OrderItem* element and another in Lines 22–25 for the *PreviousItem* element.

To translate generalization/specialization, XML Schema uses the concept of *substitution groups* to allow the use of multiple element types in a given context (in Lines 47 and 53). In our example, we generate the group *CustomerDetails* and nest the details of *Customer* such as *CustomerName*, *CustomerAddr*, and *Orders* under *CustomerDetails* as we do beginning in Line 56.

Finally, XML documents need to have a single content root node. Thus, we assume the existence of an element called *Document* (Line 4) that serves as the universal content root.

3.2.2 Translation from XML Schema to C-XML

We can convert an XML schema S to a C-XML model instance C_S by generating object sets for each element and attribute type, connected by relationship sets according to the nesting structure of S. Figure 3 shows the result of applying our proposed conversion process to the XML Schema instance of Figure 2.[2] Note that we nest object and relationship sets inside one another corresponding to the nested element structure of the XML Schema instance. Whether we display C-XML object sets inside or outside one another has no semantic significance. The nested structure, however, is convenient because it corresponds to the natural XML Schema instance structure.

The initial set of generated object and relationship sets is straightforward. Each element or attribute generates exactly one object set, and each element that is nested inside another element generates a relationship set connecting the two. Each attribute associated with an element e always generates a corresponding object set a and a relationship set r connecting a to the object set generated by e. Participation constraints for attribute-generated relationship sets are always *1..** on the a side and are either *1* or *0..1* on the e side. Participation constraints for relationship sets generated by element nesting require a bit more work. If the element is in a *sequence* or a *choice*, there may be specific minimum/maximum occurrence constraints we can use directly. For example, according to the constraints on Line 60 in Figure 2 a *CustomerDetails* element may contain a list of 0 or more *Order* elements. However, an *Order* element must be nested inside a *CustomerDetails* element. Thus, for the relationship set connecting *CustomerDetails* and *Order*, we place participation constraints of *0..** on the *CustomerDetails* side, and *1* on the *Order* side.

In order to make the generated C-XML model instance less redundant, we look for certain patterns and rewrite the generated model instance when appropriate. For example, since *ItemNr* has a key constraint, we infer that it is one-to-one with *Item*. Further, the keyref constraints on *ItemNr* for *PreviousItem* and *OrderItem* indicate that rather than create two additional *ItemNr* object sets, we can instead relate *PreviousItem* and *OrderItem* to the *ItemNr* nested in *Item*. Another optimization is the treatment of substitution groups. In our example, since *RegularCustomer* and *PreferredCustomer* are substitutable for *Customer*, we construct a generalization/specialization for the three object sets and factor out the common substructure of the specializations into the generalization. Thus, *CustomerDetails* exists in a one-to-one relationship with *Customer*.

Another complication in XML Schema is the presence of anonymous types. For example, the complex type in Line 5 of Figure 2 is a choice of 0 or more *Customer* or *Item* elements. We need a generalization/specialization to represent this, and since C-XML requires names for object sets,

[2]The particular graphical layout is human-created. Our algorithm only supplies the C-XML structure. We are considering using automated layout algorithms like AGLO, but humans generally need to adjust the output anyway.

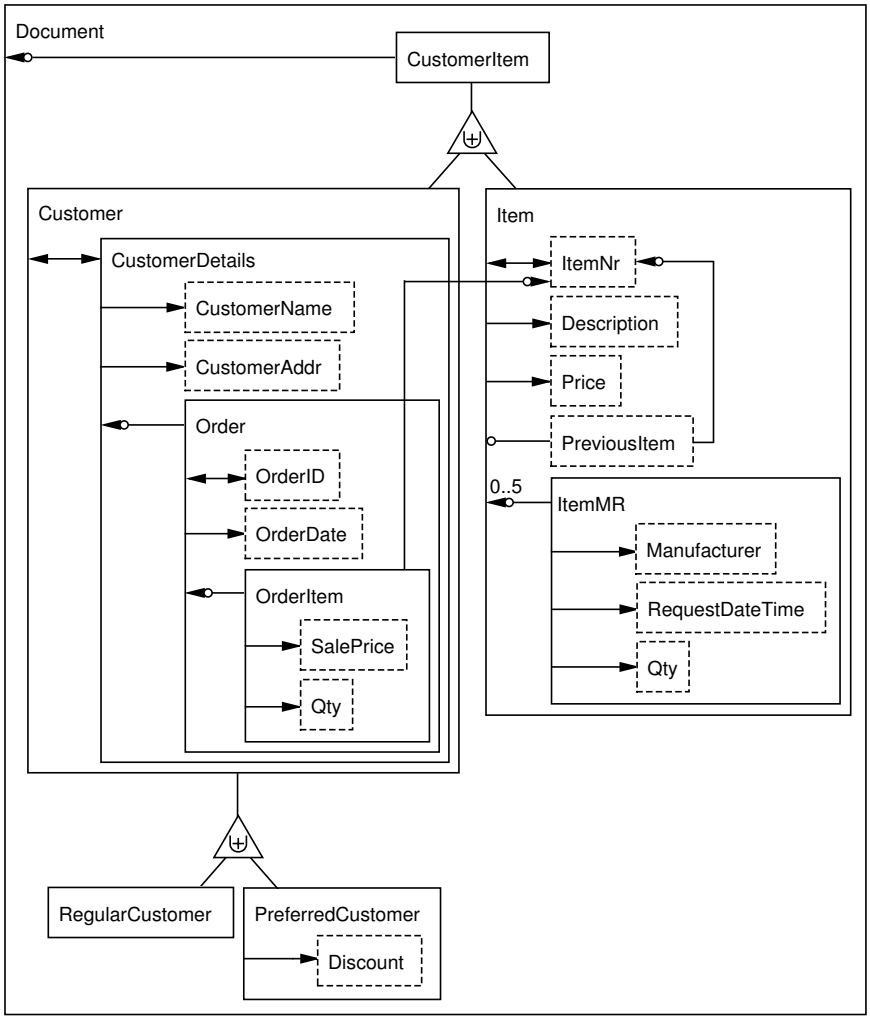

Figure 3: C-XML Model Instance Translated from XML Schema Instance of Figure 2.

we simply concatenate all the top-level names to form the generalization name *CustomerItem*.

There are striking differences between the C-XML model instances of Figures 1 and 3. The translation to XML Schema introduced new elements *Document*, *CustomerDetails*, *OrderItem*, and *ItemMR* in order to represent a top-level root node, generalization/specializations, and decomposed n-ary relationship sets. If we knew that a particular XML Schema instance was generated from an original C-XML model instance, we could perform additional optimizations. For example, if we knew *CustomerDetails* was fabricated by the translation to XML Schema, we could observe that in the reverse translation to C-XML it is superfluous because it is one-to-one with *Customer*. Similarly, we could recognize that *Document* is a fabricated top-level element and omit it from the reverse translation; this would also eliminate the need for *CustomerItem* and its generalization/specialization. Finally, we could recognize that n-ary relationship sets have been decomposed, and in the reverse translation reconstitute them. The original C-XML to XML Schema translation could easily place annotation objects in the generated XML Schema instance marking elements for

this sort of optimization.

3.3 Integration

Early work on schema integration focused on the problem of managing the data modeling and schema design process when different people on a design team were responsible for different parts of the design [BL84, EN84, NG82]. The first multidatabase system that incorporated schema integration was Multibase [Day84, NG81], which used a functional data model similar to DAPLEX [Shi82] as a common data model. Many systems were proposed or implemented with integration based on an outer join or a generalization mechanism [Alb96, RU96], as well as systems that allowed for more general ways in which objects may be related or merged [ACHK93].

Later, work on schema integration turned to work on automating schema integration and to work on data integration [RB01] surveys much of the initial work in these areas. Later work is typified by [MBR01], [DDH01], [YMHF01], [EJX02], [BE03], [DMD+03], and [XE05].

In more recent integration research an increasingly important topic is the integration of two XML repositories. Some of the research in integrating XML data sources has concentrated on schema matching [DDL00, LYHY02, RDM04]. In addition, many XML data integration systems have been developed: Agora [MFK01], SilkRoute [FMS01], XPeranto [CKS+00], MARS [DT03a, DT03b], MIX [BGL+99], YAT [CCS00] and others [WLL04, TM02, CAFO02]. In Agora [MFK01] relational and tree-structured data sources are defined as views over the XML global schema by means of an intermediate virtual relational schema closely modeling the generic structure of an XML document. Thus, the system follows a local-as-view approach [Ull97]. Also, it provides an algorithm for translating XQuery FLWR expressions into SQL. SilkRoute [FMS01] and XPeranto [CKS+00] both adopt a global-as-view [Ull97] approach and rely in a correspondence between a generic DTD and a single relational source. MARS [DT03a, DT03b] uses both global-as-view and local-as-view approaches for XML integration. MIX [BGL+99], XMF [LMP02], and YAT [CCS00] rely on the well known mediator architecture and wrap relational data sources as XML views. [WLL04] merges XML documents with different structures. [CAFO02] merges templates that specify how to recursively combine two XML documents. [TM02] proposes a semantic approach to integration of heterogeneous XML data by building a domain ontology. None of these XML integration systems, however, makes use of conceptual modeling to assist in the integration process.

In our proposal, we plan to handle integration at the conceptual level, rather than at the XML Schema level. Thus, to integrate two XML repositories each described by an XML schema we first translate each XML schema to C-XML. Then, we integrate the two C-XML model instances to create an integrated C-XML model instance. We then translate the integrated C-XML model instance back into an XML schema creating an integrated XML schema for the original XML schemas. In addition, we integrate the XML repositories at the data instance level in this same manner. The only research that we are aware of with which we can compare our approach is the

research reported in [YLL03]. The input in [YLL03] is a set of ORA-SS schema diagrams, which have been translated from XML sources. The input ORA-SS schema diagrams may also have been enriched by hand with additional necessary semantics. The output is an integrated ORA-SS schema diagram. In our approach, we translate from XML schemas rather than source documents. Thus, we can translate without manual intervention. In addition, we also translate from the integrated conceptual model instance back to an integrated XML schema.

We illustrate our proposed integration process by an example. Figures 4 and 5 represent two XML schemas that need to be integrated. Figures 6 and 7 represent two possible XML documents that correspond to Figures 4 and 5 respectively. First, we translate each XML schema to its equivalent C-XML model instance. Figures 8 and 9 show the translated C-XML model instances for the XML schemas of Figures 4 and 5 respectively.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
3: <xs:element name="Customer" abstract="true"/>
4: <xs:element name="PreferredCustomer" substitutionGroup="Customer">
5:     <xs:complexType>
6:         <xs:group ref="CustomerDetails"/>
7:         <xs:attribute name="Discount" type="xs:string" use="required"/>
8:     </xs:complexType>
9: </xs:element>
10: <xs:element name="RegularCustomer" substitutionGroup="Customer">
11:     <xs:complexType>
12:         <xs:group ref="CustomerDetails"/>
13:     </xs:complexType>
14: </xs:element>
15: <xs:group name="CustomerDetails">
16:     <xs:sequence>
17:         <xs:element name="CustomerName" type="xs:string"/>
18:         <xs:element name="CustomerAddr" type="xs:string"/>
19:         <xs:element name="Order" minOccurs="0" maxOccurs="unbounded">
20:             <xs:complexType>
21:                 <xs:attribute name="OrderID" type="xs:positiveInteger" use="required"/>
22:                 <xs:attribute name="OrderDate" type="xs:date" use="required"/>
23:             </xs:complexType>
24:         </xs:element>
25:     </xs:sequence>
26: </xs:group>
27: </xs:schema>
```

Figure 4: XML Schema1.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
3:<xs:element name="Consumer">
4:     <xs:complexType>
5:         <xs:group ref="ConsumerDetails"/>
6:     </xs:complexType>
7: <xs:group name="ConsumerDetails">
8:     <xs:sequence>
9:         <xs:element name="ID" type="xs:string"/>
10:         <xs:element name="Name" type="xs:string"/>
11:         <xs:element name="Addr" type="xs:string"/>
12:         <xs:element name="Order" minOccurs="1" maxOccurs="unbounded">
13:             <xs:complexType>
14:                 <xs:attribute name="ID-Nr" type="xs:string" use="required"/>
15:                 <xs:attribute name="Day" type="xs:date" use="required"/>
16:                 <xs:attribute name="Month" type="xs:date" use="required"/>
17:                 <xs:attribute name="Year" type="xs:date" use="required"/>
18:             </xs:complexType>
19:         </xs:element>
20:     </xs:sequence>
21: </xs:group>
22: </xs:schema>
```

Figure 5: XML Schema2.

To integrate Figures 8 and 9, we need to first obtain the mapping between the object and relationship sets in the two C-XML model instances. We plan to provide a tool or adopt a tool

```
<?xml version="1.0" encoding="UTF-8"?>
 <Customer>
    <RegularCustomer>
        <CustomerDetails>
            <CustomerName>Mary Anderson</CustomerName>
            <CustomerAddr>15 S 900 E, Provo, Utah</CustomerAddr>
            <Order OrderID="1" OrderDate="04/23/04">
            </Order>
        </CustomerDetails>
    </RegularCustomer>
    <PreferredCustomer>
        <CustomerDetails>
            <Discount>".10"</Discount>
            <CustomerName>Steve Johnson</CustomerName>
            <CustomerAddr>75 Tiger Ln, Orem, Utah</CustomerAddr>
            <Order OrderID="2" OrderDate="01/20/04">
            </Order>
        </CustomerDetails>
    </PreferredCustomer>
    <PreferredCustomer>
        <CustomerDetails>
            <Discount>".15"</Discount>
            <CustomerName>Dave King</CustomerName>
            <CustomerAddr>123 Maple Drive, Provo, Utah</CustomerAddr>
        <CustomerDetails>
    </PreferredCustomer>
</Customer>
```

Figure 6: XML1 Document for XML Schema1.

such as [MHH+01], [NM00], or [MFRW00] in which a user can specify these mappings and do the integration. (Tools for semiautomatic schema mapping such as [RB01] or [XE05] could be plugged into the tool to help a user specify mappings, but we do not consider this to be part of our work.) A user begins by declaring mappings. Assume that the mappings are as follows. For direct mappings, we have:

- *Customer* in C-XML1 = *Consumer* in C-XML2,
- *CustomerDetails* in C-XML1 = *ConsumerDetails* in C-XML2,
- *CustomerAddr* in C-XML1 = *Addr* in C-XML2,
- *Order* in C-XML1 = *Order* in C-XML2,
- *OrdedID* in C-XML1 = *ID-NR* in C-XML2.

For indirect mappings, we have:

- *OrderDate* in C-XML1 = $\pi_{Date}(\gamma_{Date:=(Month+\text{“ ”}+Day+\text{“ ”}+Year)}\pi_{Month,Day,Year}(Order—Month \bowtie Order—Day \bowtie Order—Year))$ in C-XML2 .
- *Order—Date* in C-XML1 = $\pi_{Order,Date}(\gamma_{Date:=(Month+\text{“ ”}+Day+\text{“ ”}+Year)}\pi_{Month,Day,Year}(Order—Month \bowtie Order—Day \bowtie Order—Year))$ in C-XML2.

Here, the notation $A—B$ denotes a relation between A and B, and γ is a concatenation operator. The γ operator has the form $\gamma_{B:=(A_1+...+A_n)}r$ where B is a new attribute not among the attributes of the relation r and each A_i, $1 \leq i \leq n$, is either an attribute of r or a string. The result of the γ operator is r with an additional attribute B, where each B value on row k is a concatenation of the given strings and the specified attribute values from row k. This last mapping is for a relationship

```
<?xml version="1.0" encoding="UTF-8"?>
 <Consumer>
    <ConsumerDetails>
        <ID>"6492"</ID>
        <Name>Mary Anderson</Name>
        <Addr>15 S 900 E, Provo, Utah</Addr>
        <Order>
            <ID-Nr>"1"</ID-Nr>
            <Month>"4"</Month>
            <Day>"23"</Day>
            <Year>"04"</Year>
        </Order>
    </ConsumerDetails>
    <ConsumerDetails>
        <ID>"7012"</ID>
        <Name>Larry Smith</Name>
        <Addr>1960 N 17 W, Provo, Utah</Addr>
        <Order>
            <ID-Nr>"5"</ID-Nr>
            <Month>"12"</Month>
            <Day>"10"</Day>
            <Year>"04"</Year>
        </Order>
    </ConsumerDetails>
    <ConsumerDetails>
        <ID>"1741"</ID>
        <Name>Steve Johnson</Name>
        <Addr>75 Tiger Ln, Orem, Utah</Addr>
        <Order>
            <ID-Nr>"2"</ID-Nr>
            <Month>"2"</Month>
            <Day>"2"</Day>
            <Year>"04"</Year>
        </Order>
    </ConsumerDetails>
</Consumer>
```

Figure 7: XML2 Document for XML Schema2.

set. Most mappings for relationship sets are immediate as soon as their associated object sets are matched. Thus the user only needs to specify those that are not immediate.

There are many ways to integrate the two conceptual model instances. One way is similar to a full outer join of the object and relationship sets in the two instances. The outer join merges the equivalent object sets and relationship sets from the original conceptual models. It then adds object sets and relationship sets that do not have correspondences to the integrated conceptual model instance. Another way is to let one of the conceptual model instances initially be the integrated model instance and then let the user add all the missing object sets and relationship sets from the the other conceptual model instance. A third way is to let the user choose from the two conceptual models the parts to be integrated. Ideally the tool would provide for all three ways, but as a matter of practicality, we will likely use the second way.

As the integration proceeds, conflicts can arise. Similar to [BE03], we provide a way for a user to resolve conflicts either by choosing to accept a default resolution provided by the system, or by providing whatever is necessary to resolve the conflict.

The conflicts are as follows.

- *Name conflicts.* Name conflicts arise because of synonyms or homonyms. If different names are used for the same object set, we have a synonym conflict. A default resolution for this conflict may be to let the names in one conceptual model instance be the default names for the integrated conceptual model instance. If different object sets have the same name, we have a homonym conflict. A default resolution for this conflict can be to make the object

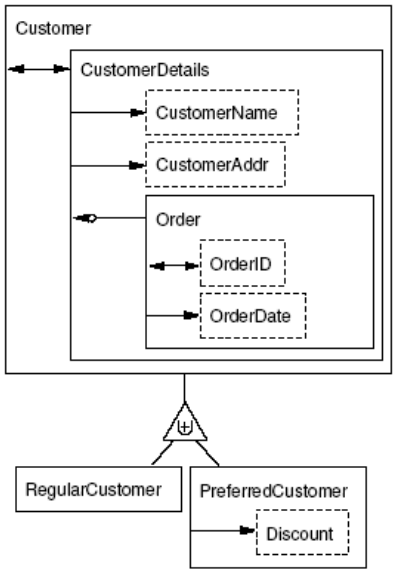

Figure 8: C-XML1 Model Instance Translated from XML Schema Instance of Figure 4.

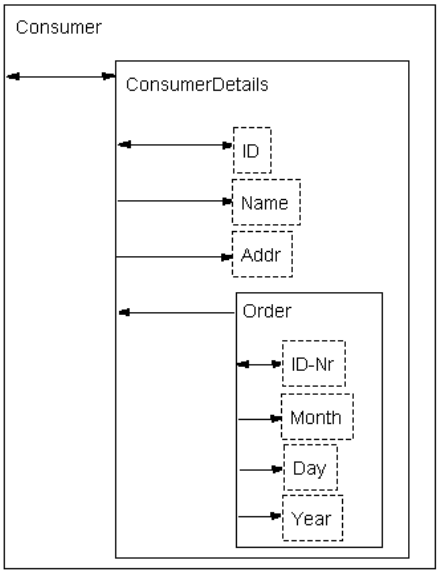

Figure 9: C-XML2 Model Instance Translated from XML Schema Instance of Figure 5.

sets have different names by adding a suffix to the names of the object sets in one of the conceptual model instances.

- *Constraint conflicts.* Constraint conflicts arise when there are inconsistencies in constraints. A default resolution of this conflict can be to loosen the constraints in the integrated conceptual model instance. For example, if the participation constraint is optional in one instance and mandatory in the other, then the integrated conceptual model instance has an optional participation constraint.

- *Structural conflicts.* Structural conflicts arise because different types of structures represent the same concept. An example is if the date in one conceptual model instance is represented in one object set as in C-XML1 (Figure 8) and is represented in another conceptual model instance in three object sets as in C-XML2 (Figure 9). The default resolution of this conflict can be to choose both resulting in an explicit aggregation of the information in the integrated conceptual model. For our example, the default would be *OrderDate* as an aggregate with *Month*, *Day*, and *Year* aggregated in the inside. For other structural conflicts, we also need to provide resolutions (e.g. those suggested in [BE03]).

- *Data type conflicts.* Data type conflicts occur when two corresponding object sets have different types, for example, *String* and *Integer*. A default resolution of this conflict can be to let the types in one conceptual model instance be the default types for the integrated model instance. In addition, the types need conversion routines that can convert between the different types.

Figure 10 represents an integrated C-XML instance for our running example in Figures 8 and 9. The integrated conceptual model instance contains representations for all the object sets and all the relationship sets in the both C-XML1 and C-XML2. When object sets are in one conceptual model instance but not in the other, we often have to loosen constraints when we integrate them. Since there are no *ID* numbers in C-XML1, they must be optional in the integrated model instance as Figure 10 shows. Similarly, since the *RegularCustomer* and *PreferedCustomer* in C-XML are (presumably) not all the customers we must drop the union constraint. We resolve the synonym conflicts by letting the system use the names in C-XML1 as default names in the integrated C-XML instance. A constraint conflict appears because *CustomerDetails* in the relationship set *CustomerDetails—Order* is optional in C-XML1 and is mandatory in C-XML2. The conflict is resolved by loosening the constraint in the integrated C-XML model instance and letting the participation constraint be optional. A structural conflict appears in the representation of the date—as one object set *OrderDate* in C-XML1 and as three object sets *Day*, *Month*, and *Year* in C-XML2. The conflict is resolved by making *OrderDate* as an aggregate with *Month*, *Day*, and *Year* aggregated in the inside. A data type conflict appears if *OrderID* is a *PositiveInteger* in C-XML1

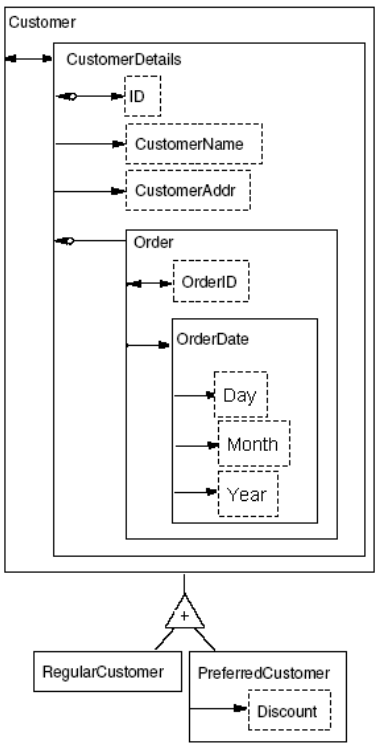

Figure 10: Integrated C-XML for Figures 8 and 9.

and is a string in C-XML2. The conflict can be resolved by choosing the type *PositiveInteger* in the integrated C-XML instance. After obtaining the integrated C-XML model instance, we can use the translation procedure described in Section 3.2.1 to create the XML schema for the result. For our example, we translate the integrated C-XML model instance to the XML schema in Figure 11 which represents the integrated XML schema for the two original XML schemas.

Concerning the merging of data, we merge lexical objects if their values are identical. Sometimes, we must convert values before comparing them. When conversion is necessary, we can either convert both sets of values from the two object sets which are to be merged into a common form or convert the form of one set of values to the form of the other. There are three types of data conflicts.

- *Format conflicts.* Format conflicts arise when different formats are used for the same data instance. An example is the format of dates: "12-01-05" versus "01/12/05" for January 12, 2005. As a default resolution, we can convert both to a standard format. We can, for example, convert both to "12 Jan 2005".

- *Units conflicts.* Unit conflicts arise differences in the choice of units of measurement for numerical data. An example of a unit conflict is to represent the length of a line of printed text in inches in one XML document and in centimeters in another XML document. The default resolution of this conflict can be to let the units in one XML document be the units

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
3: <xs:element name="Customer" abstract="true"/>
4: <xs:element name="PreferredCustomer" substitutionGroup="Customer">
5:     <xs:complexType>
6:         <xs:group ref="CustomerDetails"/>
7:         <xs:attribute name="Discount" type="xs:string" use="required"/>
8:     </xs:complexType>
9: </xs:element>
10: <xs:element name="RegularCustomer" substitutionGroup="Customer">
11:     <xs:complexType>
12:         <xs:group ref="CustomerDetails"/>
13:     </xs:complexType>
14: </xs:element>
15: <xs:group name="CustomerDetails">
16:     <xs:sequence>
17:         <xs:element name="ID" type="xs:string"/>
18:         <xs:element name="CustomerName" type="xs:string"/>
19:         <xs:element name="CustomerAddr" type="xs:string"/>
20:         <xs:element name="Order" minOccurs="0" maxOccurs="unbounded">
21:             <xs:complexType>
22:                 <xs:attribute name="OrderID" type="xs:positiveInteger" use="required"/>
23:                 <xs:element name="OrderDate">
24:                     <xs:complexType>
25:                         <xs:attribute name="Day" type="xs:date" use="required"/>
26:                         <xs:attribute name="Month" type="xs:date" use="required"/>
27:                         <xs:attribute name="Year" type="xs:date" use="required"/>
28:                     </xs:complexType>
29:                 </xs:element>
30:             </xs:complexType>
31:         </xs:element>
32:     </xs:sequence>
33: </xs:group>
34: </xs:schema>
```

Figure 11: Integrated XML Schema.

for the integrated XML document. We, of course, need to provide conversion routines that convert between the different units.

- *Arbitrary lexical identifier conflicts.* We would not normally expect arbitrary identifiers, such as employee ID numbers or bin numbers, to be the same for different organizations whose XML repositories are to be merged. In our example, *OrderID*'s are arbitrary lexical identifiers. As a default resolution, we can prefix these identifiers with a designator that specifies the repository from which they originated.

To merge nonlexical objects, we assign new object identifiers to all objects. We assign the same object identifier to objects if there is sufficient evidence from their associated lexical values to indicate identity. A user can declare which related values must match to have the same identity. As a default we can assume identifying key values if they exist.

Figure 12 shows the integrated XML document for the XML documents in Figures 6 and 7. In our example, we assume that a user has declared that *Customer* objects are the same if the *CustomerAddr* and *CustomerName* are the same. We also assume that the user has specified that *OrderID*'s are to have a prefix—"1-" if from C-XML1 and "2-" if from C-XML2. Finally, we assume that all dates are to be converted to a system standard date of the form two-digit day, three-letter names, four-digit year.

```
<?xml version="1.0" encoding="UTF-8"?>
<Customer>
    <RegularCustomer>
        <CustomerDetails>
            <CustomerID>"6492"</CustomerID>
            <CustomerName>Mary Anderson</CustomerName>
            <CustomerAddr>15 S 900 E, Provo, Utah</CustomerAddr>
            <Order>
                <OrderID>"1-1"</OrderID>
                <OrderDate>
                    <Day>"23"</Day>
                    <Month>Apr</Month>
                    <Year>"2004"</Year>
                </OrderDate>
            </Order>
            <Order>
                <OrderID>"2-1"</OrderID>
                <OrderDate>
                    <Day>"23"</Day>
                    <Month>Apr</Month>
                    <Year>"2004"</Year>
                </OrderDate>
            </Order>
        </CustomerDetails>
    </RegularCustomer>
    <PreferredCustomer>
        <CustomerDetails>
            <Discount>".10"</Discount>
            <CustomerID>"1741"</CustomerID>
            <CustomerName>Steve Johnson</CustomerName>
            <CustomerAddr>75 Tiger Ln, Orem, Utah</CustomerAddr>
            <Order>
                <OrderID>"1-2"</OrderID>
                 <OrderDate>
                    <Day>"20"</Day>
                    <Month>Jan</Month>
                    <Year>"2004"</Year>
                </OrderDate>
            </Order>
            <Order>
                <OrderID>"2-2"</OrderID>
                <OrderDate>
                    <Day>"02"</Day>
                    <Month>Feb</Month>
                    <Year>"2004"</Year>
                </OrderDate>
            </Order>
        </CustomerDetails>
    </PreferredCustomer>
    <PreferredCustomer>
        <CustomerDetails>
            <Discount>".15"</Discount>
            <CustomerName>Dave King</CustomerName>
            <CustomerAddr>123 Maple Drive, Provo, Utah</CustomerAddr>
        </CustomerDetails>
    </PreferredCustomer>
    <RegularCustomer>
        <CustomerDetails>
            <CustomerName>Larry Smith</CustomerName>
            <CustomerAddr>1960 N 17 W, Provo, Utah</CustomerAddr>
            <Order>
                <OrderID>"5"</OrderID>
                <OrderDate>
                    <Day>"12"</Day>
                    <Month>Oct</Month>
                    <Year>"2004"</Year>
                </OrderDate>
            </Order>
        <CustomerDetails>
    </RegularCustomer>
</Customer>
```

Figure 12: Integrated XML Document.

4 Research Plan

4.1 Approach

We plan to build a prototype that translates from C-XML to XML Schema and vice versa. In addition we plan to build a prototype that allows a user to integrate two XML schemas at the conceptual level.

To show that the implemented algorithms have the properties we expect them to have, we will do proofs. We will formalize C-XML using first-order predicate calculus and provide definitions, lemmas, and theorems to be proven. In more detail we plan to prove the following lemmas and theorems.

Lemma 1 *Let $I_{S_{C-XML}}$ be a valid interpretation for a populated C-XML model instance S_{C-XML}. There exists a translation t_{C-XML} that correctly represents $I_{S_{C-XML}}$ as a valid interpretation $I_{S^{PC}_{C-XML}}$ in predicate calculus.*

Lemma 2 *Let $I_{S_{XMLSchema}}$ be an XML document that conforms to an XML Schema instance $S_{XMLSchema}$. There exists a translation $t_{XMLSchema}$ that correctly represents $I_{S_{XMLSchema}}$ as a valid interpretation $I_{S^{PC}_{XMLSchema}}$ in predicate calculus.*

Theorem 1 *Let T be the translation that translates a C-XML model instance S_{C-XML} to an XML Schema instance $S_{XMLSchema}$. T preserves information and constraints.*

Theorem 2 *Let T be the translation that translates an XML Schema instance $S_{XMLSchema}$ to a C-XML model instance S_{C-XML}. T preserves information and constraints.*

The following two theorems will need a careful enumeration of assumptions. These assumptions will correspond with the default assumptions of our integration system. The theorems will also need to rely on carefully written definitions. The basic ideas, however, are straightforward, and we state them in this slightly loose but straightforward way.

Theorem 3 *The integrated C-XML model instance can include all concepts in both C-XML source instances.*

Theorem 4 *The integrated XML document can include all data in both XML source documents.*

4.2 Artifacts to be Produced

A prototype will be produced that: (1) automatically translates C-XML to XML Schema, (2) automatically translates XML Schema to C-XML, and (3) allows a user to integrate two XML schemas via C-XML views.

4.3 Limitations of the Dissertation

The following are limitations of this dissertation.

- The prototype will not generate automatic mappings for schema integration. (The mappings will instead be user specified and will be consistent with those automatically generated so that automatic mapping generators such as [Xu03] can be accommodated. Similarly object identity, type, unit, and format conflicts will be resolved by the user.)
- Development of the prototype will not include development of sophisticated C-XML automatic layout tools. (Existing layout tools may be adopted or a naive approach may be implemented.)
- There will be no claims or proof of claims that the prototype is easy to use. (The prototypes should be reasonably usable, but no attempt will be made to verify ease of use through controlled user experiments.)

5 Research Papers

The following are proposed titles of papers that could be written based on the results of the research.

- An Information- and Constraint-Preserving Translation from C-XML to XML Schema
- An Information- and Constraint-Preserving Translation from XML Schema to C-XML
- On the Correctness of Conceptual Level Integration of XML Schemas
- C-XML-I: A Tool for Conceptual Level Integration of XML Schemas

6 Contribution to Computer Science

We will have offered Conceptual-XML (C-XML) as an answer to the challenge of systems analysis for XML repositories. We will have shown that C-XML is equivalent in expressive power to XML Schema. Along with C-XML, we will have provided and proved lemmas and theorems that translations between C-XML and XML Schema preserve information and constraints. We will also have shown how to handle heterogeneity by dealing with it at the conceptual level which is likely to be much better than dealing with it at the level of an XML schema. Further, we will have provided lemmas and theorems to show that the conceptual-level integration indeed properly provides for XML schema integration.

7 Dissertation Schedule

The following is a proposed schedule for completing the work.

- Chapter 1 Introduction (July 2006-August 2006)
- Chapter 2 Translations between C-XML and XML Schema
  - Translate from C-XML to XML Schema (February 2005 - April 2005)
  - Translate from XML Schema to C-XML (May 2005 - July 2005)
- Chapter 3 Integration
  - Tool Development (August 2005 - April 2006)
  - Correctness Proof (May 2005 - June 2006)
- Chapter 4 Conclusions September 2006

8 References

References

[ACHK93] Y. Arens, C.Y. Chee, Chun-Nan Hsu, and C.A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1993.

[Alb96] J. Albert. Data integration in the RODIN multidatabase system. In *Proceedings of the 1st International Conference on Cooperative Information Systems (CoopIS1996)*, page 48, Brussels, Belgium, June 1996.

[BE03] J. Biskup and D.W. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Information Systems*, 28(3):169–212, 2003.

[BEA] BEA systems, inc. www.bea.com.

[BGH00] L. Bird, A. Goodchild, and T. Halpin. Object role modelling and XML-schema. In *Proceedings of the Ninteenth International Conference on Conceptual Modeling (ER2000)*, pages 309–322, Salt Lake City, Utah, October 2000.

[BGL+99] C.K. Baru, A. Gupta, B. Ludscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-based information mediation with MIX. *ACM SIGMOD Record*, 28(2), June 1999.

[BL84] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, 10(6):650–664, October 1984.

[CAFO02] S. Castano, V.D. Antonellis, A. Ferrara, and G.S.K. Ottathycal. Ontology-based integration of heterogeneous XML datasources. In *Proceedings of the 10th Italian Symposium on Advanced Database Systems (SEBD2002)*, pages 27–41, Isola d'Elba, Italy, June 2002.

[CCS00] V. Christophides, S. Cluet, and J. Simèon. On wrapping query languages and efficient XML integration. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pages 141–152, Dallas, Texas, May 2000.

[CKS+00] M.J. Carey, J. Kiernan, J. Shanmugasundaram, E.J. Shekita, and S.N. Subramanian. XPERANTO: A middleware for publishing object-relational data as XML documents. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB2000)*, pages 646–648, Cairo, Egypt, September 2000.

[CLL02] Y.B. Chen, T.W. Ling, and M.L. Lee. Designing valid xml views. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER2002)*, pages 463–477, Tampere, Finland, October 2002.

[CLL03] Y.B. Chen, T.W. Ling, and M.L. Lee. Automatic generation of XQuery view definitions from ORA-SS views. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER2003)*, pages 158–171, Chicago, Illinois, October 2003.

[CSF00] R. Conrad, D. Scheffner, and J.C. Freytag. XML conceptual modeling using UML. In *Proceedings of the Ninteenth International Conference on Conceptual Modeling (ER2000)*, pages 558–571, Salt Lake City, Utah, October 2000.

[Day84] U. Dayal. View definition and genarlization for database integration in a multidatabase system. *IEEE Transactions on Software Engineering*, 10(6):628–644, September 1984.

[DDH01] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, pages 509–520, Santa Barbara, California, May 2001.

[DDL00] A. Doan, P. Domingos, and A.Y. Levy. Learning source description for data integration. In *Proceedings of the International Workshop on The Web and Databases (WebDB2000)*, pages 81–86, 2000.

[dH01] R. dos Santos Mello and C.A. Heuser. A rule-based conversion of a DTD to a conceptual schema. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER2001)*, pages 134–148, Yokohama, Japan, November 2001.

[DMD+03] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. *VLDB Journal*, 12:303–319, 2003.

[DT03a] A. Deutsch and V. Tannen. MARS: A system for publishing XML from mixed and redundant storage. In *Proceedings of the International Conference on Very Large Databases (VLDB2003)*, pages 201–212, Berlin, Germany, September 2003.

[DT03b] A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In *Proceedings of the 9th International Conference on Database Theory (ICDT2003)*, pages 255–241, Siena, Italy, January 2003.

[DWLL00] G. Dobbie, X.Y. Wu, T.W. Ling, and M.L. Lee. ORA-SS: An object-relationship-attribute model for semistructured data. Technical report TR 21/00, School of Computing, National University of Singapore, 2000.

[EJX02] D.W. Embley, D. Jackman, and L. Xu. Attribute match discovery in information integration: Exploiting multiple facets of metadata. *Journal of the Brazilian Computing Society*, 8(2):32–43, November 2002.

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[EM01] D.W. Embley and W.Y. Mok. Developing XML documents with guaranteed 'good' properties. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER2001)*, pages 426–441, Yokohama, Japan, November 2001.

[EN84] R. Elmasri and S.B. Navathe. Object integration in logical database design. In *Proceedings of the First International Conference on Data Engineering*, pages 426–433, Washington, DC, April 1984.

[EWH+02] R. Elmazri, Y.-C. Wu, B. Hojabri, C. Li, and J. Fu. Conceptual modeling for customized XML schemas. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER2002)*, pages 429–443, Tampere, Finland, October 2002.

[FMS01] M.F. Fernandez, A. Morishima, and D. Suciu. Efficient evaluation of XML middleware queries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD2001)*, pages 103–114, Santa Barbara, California, May 2001.

[LEW00] S.W. Liddle, D.W. Embley, and S.N. Woodfield. An active, object-oriented, model-equivalent programming language. In M.P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Data Modeling*, pages 333–361. MIT Press, Cambridge, Massachusetts, 2000.

[LMP02] K. Lee, J. Min, and K. Park. A design and implementation of XML-based mediation framework (XMF) for integration of internet information resources. In *Proceedings of of the 35th Annual Hawaii International Conference on System Sciences (HICSS2002)*, pages 2700–2708, Big Island, Hawaii, January 2002.

[LYHY02] M.L. Lee, L.H. Yang, W. Hsu, and X. Yang. XClust: Clustering XML schemas for effective integration. In *Proceedings of the 11th ACM International Conference on Information and Knowledge Management (CIKM2002)*, pages 292–299, McLean, Virginia, November 2002.

[MBR01] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB2001)*, pages 49–58, Roma, Italy, September 2001.

[MFK01] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries over heterogeneous data sources. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB2001)*, pages 241–250, Roma, Italy, September 2001.

[MFRW00] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera ontology environment. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI2000)*, pages 1123–1124, Austin, Texas, 30 July-3 August 2000.

[MHH+01] R.J. Miller, M.A. Hernandez, L.M. Haas, L. Yan, C.T. Howard Ho, R. Fagin, and L. Popa. The Clio project: Managing heterogeneity. *ACM SIGMOD Record*, 30(1):78–83, 2001.

[MLM01] M. Mani, D. Lee, and R.R. Muntz. Semantic data modeling using XML schemas. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER2001)*, pages 149–163, Yokohama, Japan, November 2001.

[NG81] S.B. Navathe and S.G. Gadgil. Multibase: Integrating heterogeneous distributed database systems. In *Proceedings of the National Computer Conference*, pages 487–499, Montvale, New Jersey, May 1981.

[NG82] S.B. Navathe and S.G. Gadgil. A methodology for view inegration in logical database design. In *Proceedings of the 8th International Conference on Very Large Data Bases*, pages 142–164, Mexico City, September 1982.

[NM00] N. Noy and M. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI2000)*, pages 450–455, Austin, Texas, 30 July-3 August 2000.

[RB01] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001.

[RDM04] E. Rahm, H. Do, and S. Massmann. Matching large XML schemas. *ACM SIGMOD Record*, 33(4):26–31, December 2004.

[RU96] A. Rajaraman and J.D. Ullman. Integrating information by outerjoins and full disjunction. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems (PODS1996)*, pages 238–248, Montreal, Canada, June 1996.

[Shi82] D.W. Shipman. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1982.

[TM02] K. Tufte and D. Maier. Merge as a lattice-join of xml documents. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB2002)*, Hong Kong, China, August 2002.

[Ull97] J.D. Ullman. Information integration using logical views. In F.N. Afrati and P. Kolaitis, editors, *Proceedings of the 6th International Conference on Database Theory (ICDT1997)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40, Delphi, Greece, January 1997. Springer-Verlag.

[WLL04] W. Wei, M. Liu, and S. Li. Merging of XML documents. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004)*, pages 273–285, Shanghai, China, November 2004.

[XE05] L. Xu and D.W. Embley. A composite approach to automating direct and indirect schema mappings. *Information Systems*, 2005. (to appear).

[XML01] XML Schema Part 0: Primer: W3C Recommendation, May 2001. URL: http://www.w3.org/TR/xmlschema-0/.

[Xu03] L. Xu. *Source Discovery and Schema Mapping for Data Integration.* PhD thesis, Brigham Young University, August 2003.

[YLL03] X. Yang, M.L. Lee, and T.W. Ling. Resolving structural conflicts in the integration of XML schemas: A semantic approach. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER2003)*, pages 520–533, Chicago, Illinois, November 2003.

[YMHF01] L.L. Yan, R. Miller, L.M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. *SIGMOD Record*, 30(2):485–496, 2001.

This dissertation proposal by Reema Al-Kamha is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the dissertation proposal requirement in the Department of Computer Science for the degree of Doctor of Philosophy.

David W. Embley, Committee Chair

Stephen W. Liddle, Committee Member

Scott N. Woodfield, Committee Member

Parris Egbert, Committee Member

Eric G. Mercer, Committee Member

David W. Embley, Graduate Coordinator